Evolution of Learning Rules for Supervised Tasks I: Simple Learning Problems

Ibrahim KUSCU
Cognitive and Computing Sciences
University of Sussex
Brighton BN1 9QH
Email: ibrahim@cogs.susx.ac.uk

November 10, 1995

Abstract

Initial experiments with a genetic based encoding schema are presented as a potentially powerful tool to discover learning rules by means of evolution. Several simple supervised learning tasks are tested. The results indicate the potential of the encoding schema to discover learning rules for more complex and larger learning problems.

Keywords:Evolution, Genetic Algorithms, Supervised Learning

1 Introduction

Evolution and learning address two different levels of adaptation processes: one taking place during the life time of an organism and the other taking place during the evolutionary history of population of organisms [3] [18].

There are quite a number of research concentrating on the relationship between evolution and learning [2] [16] [11] [19] [1] [17]. The nature of interaction between the two has been shown to be complementary : the presence of learning can facilitate the process of evolution and evolutionary methods can significantly speed up the learning process.

However, in these studies, the methods of learning have been chosen beforehand and frequently back-propagation algorithm is used. Evolution is used as a mean of creating an initial state upon which learning process is applied for increased performance. However, the nature of learning during an evolutionary process has not been investigated.

An interesting experiment was carried out by Chalmers [3]. His primary aim was to observe how learning might evolve during the process of evolution. Rather than looking at the interaction between evolution and learning, the major focus of his study was evolution of learning mechanisms themselves. Starting from a random population of genome coding for the weight-space dynamics of artificial neural networks, he found out that a learning rule which is very close to the delta rule has evolved successfully in learning eight linearly separable tasks of supervised learning. A similar experiment is applied in the area of unsupervised learning to find rules for the Self-Organising Map [4]. They have supported the idea that genetic algorithms can be used to search for optimal learning algorithms by providing evidence for the existence of several learning algorithms that produced an organisation similar to that of Kohonen Algorithm. These experiments encouraged the hope that it is not impossible for such genetic based systems to discover new learning algorithms. However, realising this hope is not easy. For example, trying different supervised learning problems at a larger scale or trying another class of learning such as reinforcement learning would extensively increase the complexity of the problem. Besides, a fundamental problem in attaining such goal is the problem of genetic encoding (i.e. representation). How can we code for such large and complex dynamics? How and what prior knowledge should be introduced to reduce the scale and the complexity of the space?

The way a genome is represented has a significant affect on the way Genetic Algorithms (GA)[12] perform in finding the solution to a particular problem. GA works *directly* on the representation of the problem encoded on genome. Thus, the search space within which the possible solution lies is *directly* influenced by the representation. It is widely recognised that fixed length character strings pose severe limitations for the solution of particular problems [5] [6] Several representation schema which improve the capabilities of Genetic Algorithms have been presented by Smith [21] Koza [13] and Harvey [8][9][10]. Smith has provided an earlier example of using variable length strings in representing the genome. Harvey has shown that adaptation of variable length genome improves conventional Genetic Algorithms by changing their finite search space to an open ended search. The new form of representation called SAGA (Species Adaptation Genetic Algorithm) allows incremental evolution of genome from simple to more complex. Finally, Koza argues that fixed length representation impose difficulties particularly for those problems which require hierarchical solutions and especially when "the size and the shape" of the solution is unknown in advance. He presents Genetic Programming paradigm where general and hierarchical computer programs of varying size and shape are evolved to solve increasingly difficult problems in Artificial Intelligence. In this paradigm the genome are compositions of set of functions and terminal units *appropriate to* particular problem domain. The set of functions includes among others a set of domain specific functions. Together with terminal units, which includes inputs (sensors), the domain specific functions used in encoding genome introduces

prior-knowledge (domain-specific knowledge) to the representation of the problem. Although this reduces the scale and the complexity of the search space, it also effectively introduces possible low level solutions to the problem in hand. This form of human intervention makes it less attractive for a learning paradigm. This issue is extensively discussed in [15].

In this paper, as part of an ongoing research an encoding schema will be presented. Combined with genetic algorithms it can successfully produce evolution of learning rules. Rather than searching for a general learning algorithm (as in the work of Chalmers), the aim is to see whether evolution would produce a specific learning rule for the problem in hand. Although the representation schema is very similar to that of Koza's Genetic Programming [13], introducing prior knowledge into representation of the problem will be minimal, if any at all. In this strategy potential learning rules are encoded as random mathematical expressions at variable lengths. Mathematical expressions are used in simulating the evolution of cooperation in iterated prisoner's dilemma game by Kuscu [14]. The expressions are made up of random numbers and random variables. The variables are to be instantiated to input values of training set in a typical supervised learning. By using LISP's "EVAL" statement, the expressions are evaluated to certain numbers. This is used as a base of determining their fitness during the course of genetic based evolution to solve some learning problems. In the next section I will describe the representation strategy and the process of applying genetic algorithms. Then the result of the experiments will be presented. Finally, I will conclude with a discussion and future research possibilities using the genetic based encoding schema.

2 The Model

2.1 The Encoding Schema

The potential learning rules are encoded as simple mathematical expressions rather than bit representation. They are at variable lengths. The expressions are produced randomly involving random numbers (in some experiments real numbers and in others integers or the combination of the two has been tried) and a number of variables to be instantiated to the values of inputs from each pattern in the training set. The mathematical operators include plus, minus and multiplication (In addition to these MOD and division operators are also tried. Although their absence for the experiments to be described here did not show any noticeable difference, it reduced significantly computational cost of processing individuals). A typical expression for a problem with two input values would look like this:

```
(((1 + *I1*) + (*I2* * *I1*)) - (( 0  - *I2*) - (*I1* * *I2*)))
```

This expression is randomly produced for a problem with two input values. *I1* and *I2* are the variables to be instantiated to the input values from the patterns at each time of evaluation.

When generating the expressions a variable parameter called *percentage* is used to impose how complex we want the expressions (i.e. longness or shortness of the expressions). It can have values from 0 to 100. The higher the percentage value the more complex the expression tends to be. In the experiments variable *percentage* values are used depending on the complexity of the problem (in the range of 75 to 85).

Internally each of the expressions are represented as threes. This structure is used as a basis of applying genetic operators: crossover and mutation. A random point in selected expression (tree) is chosen as crossover or mutation point. More details will be given about this later. The typical structure of an expression would look like as in Figure 1.

```
The expression:

    (*I1* - ((*I2* + 1) * 0))

The tree representation:
```

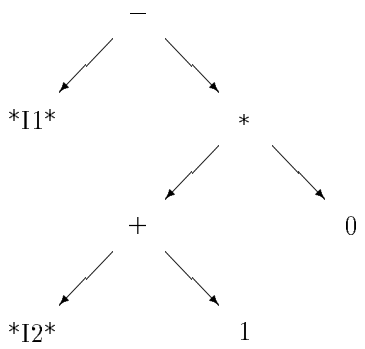

Figure 1: Tree representation of an expression.

In order to balance the behavior of the expressions (i.e. the bias toward positive expressions) half of the expressions are given a minus sign in front of

them. This is to achieve, potentially an equal chance of producing negative and positive values when generating the expressions in the initial population.

2.2 Genetic Algorithms

The Schema Theorem developed by Holland[12] based on genetic search has been proven to be useful in many applications involving large, complex and deceptive search spaces [7]. So genetic search is most likely to allow fast, robust evolution of genotypes encoding for potential learning rules as mathematical expressions. Using Genetic Algorithms (GA) the model is implemented in LISP. The top level structure of the system exhibit the following:

1. Initialize the population of expressions
2. Evaluate each expression and determine its fitness
3. Select expressions to reproduce more
4. Apply genetic operators to create new population
5. If the solution found or sufficient number of generations are created then stop; if not go to 2.

The initialization technique is randomly generating mathematical expressions. This introduces the least amount of domain specific knowledge into the initial population through the variables used in the expressions. Unlike Koza's genetic programming applied to particular problems there are no domain specific functions. Only three mathematical functions are allowed; addition, subtraction and multiplication.

In the following sections, I will describe the rest of the steps in applying GA.

2.2.1 Evaluation

In order to provide a basis to determine the fitness of the expressions, each generation the expressions are evaluated using Lisp's "EVAL" statement by instantiating input values for each of the patterns from the training set.

The fitness of an expression is based on its success in learning a specific task. Since the target outputs are in the range of 0 to 1, the values, once obtained after the evaluation of the expressions, are mapped to values between 0 and 1 by using a squashing function. Several functions have been tested in this mapping including logistic activation function used by [20]. One of the functions which showed the most success, especially in mapping to binary target outputs, was the following:

```
if value > 1 return 1
if value < 0 return 0
otherwise return the value
```

The fitness (success) of the individual expression is computed by testing them on all training patterns, and dividing the total error by the number of patterns, subtracting from 1 and multiplying by 100 yielding a fitness percentage between 0 and 100.

The expressions are ranked after each generation according to their success. Those who are higher in the rank (higher scoring ones) are said to be most fitting expressions.

2.2.2 Selection

The purpose of selection in GA is to give better opportunity of reproducing to those members of the population which shows better fitness. For the model this means to select those expressions with higher scores (beginning part of the rank) and give them more chance to reproduce.

In the model, parent selection technique for reproduction is normalizing by using an exponential function taken from Whitley's [23] rank-based selection technique. The function generates integer numbers from 1 to population size. The generation of numbers exhibits characteristics of a non-linear function where there is more tendency to produce smaller numbers (since higher scoring expressions are on top of the rank).

The function is $Z = X - \sqrt{\frac{X*X-4*(X-1)*Y}{2*(X-1)}}$ The selection algorithm is based on the X, Y, Z values in the above formula where X is a bias computed as $1+Y$ where Y is a random number between 0 and 1. The value of Z lies between 0 and 1 and in the rank-ordered population N the expression at position $N*Z$ is chosen.

The number produced by this function is used as an index to the ranked population of expressions from highest scoring ones to the lowest scoring. Then, after producing two indices by using the selection function the corresponding expressions are selected to undergo the genetic operators.

2.2.3 Genetic Operators

Applying genetic operators introduces variation to the population of expressions and allows the components (genes) of better performing expressions to live longer. This creates the necessary environment to cause evolution. In order to accomplish this it uses two different genetic operators; crossover and mutation. However, implementation of the both of the operators used by the system are different than conventional implementations on bit strings since the length of expressions are variable.

In order to apply genetic operators two parent expressions are selected by using the selection function. The crossover algorithm (one point) requires that a point should be selected on each parent at random dividing parents into two. Then the corresponding parts of the parents are exchanged producing two different children. The internal representation of expressions in the system are

binary tree representations. In order to choose a point in this tree two different probabilities are used. One probability determines whether we want to go to the left or right branches of the tree and the other determines whether to go down more or to stay at that level. Figure 2 shows systematically how these are implemented in the system.

```
if mutate
   then create new expression
else
   if at end node of either tree
      or probability-down > cutoff
      then swap parts of trees
   else
      if probability-left > cutoff
         then go down on the left branch
              and recurse
      else
         go down on the right branch
         and recurse
```

Figure 2: Crossover and mutation algorithm.

When the point is chosen, the next thing to decide is whether there will be a mutation. If there will be a mutation on both of the trees at that point a new expression is added. Otherwise the parts of the trees side apart from that point are swapped.

3 Results

In the preliminary experiments of the model, the emphasis has been on simplicity to keep the computational requirements manageable whilst encouraging the achievement of meaningful results.

The model is applied to several supervised learning tasks involving varying difficulty of input-output mappings. The aim of the experiments is to observe whether a learning rule would emerge during the course of evolution for a specified task. First experiments involved OR, AND and XOR problems.

3.1 OR, AND and XOR problems

Although, a learning rule would emerge easily in one or two generations for OR or AND problems, it took longer to discover a learning rule for XOR problem. Here are the typical solutions found for OR and AND problems.

```
LEARNING RULES FOR THE "AND" PROBLEM

1.  (*I2* * *I1*)

2.  ((0 * *I2*) + (*I1* * *I2*))

3.  ((*I1* * *I2*) - (1 - *I1*))

4.  (((*I1* * *I2*) + (*I2* + *I2*)) - (*I2* + *I2*))
```

Note that the above rules are either exact mathematical representation (i.e. rule 1) of 'AND' logical operator or can be simplified to it in accordance with the squashing function used.

```
LEARNING RULES FOR THE "OR" PROBLEM

1.  (*I2* + *I1*)

2.  (((*I2* * *I1*) + (*I2* + *I2*))
    +
    ((((*I2* * *I1*) - (1 - *I1*)) + (1 + *I1*)) + (*I2* - *I2*)))

3.  ((0 * *I1*) + ((*I1* + *I1*) - ((*I1* - *I2*) - (*I2* * *I1*))))
```

The rules evolved for the 'OR' problem also show the same properties as 'AND' problem. The first rule is exact mathematical representation of the 'OR' rule taking into consideration the fitness function which evolve to 1 when both inputs are 1 but the expression produces 2. Similarly other rules exhibit different ways of expressing the solution to the problem.

The following are the expressions which evolved to learn logical XOR problem. The degree of difficulty for a learning rule to code for XOR problem increased drastically compared to AND and OR problems. An expression which codes a learning rule for the AND problem and OR problem would evolve in a couple of generations. However, it took at least 10 generations with 20 population numbers to evolve for the XOR problem. The parameters for the genetic

algorithms for these experiments were 10 percent mutation rate and 50 percent crossover rate. The success of the evolved learning rules were 100 percent.

LEARNING RULES FOR THE XOR PROBLEM

```
1.   (- 1 (+ *I1* (- *I2* (* *I2* (+ *I1* *I1*))))))

2.   (((1 - (0 + *I2*))

     + (((*I2* + ((*I2* - 0) * 1)) * *I1*) - *I1*))))

3.   (((((0 - *I2*) + (((*I2* - *I2*) - (*I2* - *I1*))
     +
     (1 + *I1*))) - ((*I2* + *I1*)
     -
     ((1 - *I2*) + (((*I2* + *I1*) - (0 * *I1*)) - (0 - *I2*)))))
     *
     ((1 + *I2*) - (*I1* - *I2*)))


4.   (((((*I2* * *I1*) + ((*I1* * *I2*) - (*I1* - *I2*)))
     +
     ((1 * *I1*) - (*I2* + *I1*))) + (1 - *I2*))


5.   (((((((1 + *I1*) + (*I2* * *I1*)) - ((*I1* - *I2*)
     -
     (*I1* * *I2*))) + (0 * *I2*))
     +
     (((*I2* + *I1*) - ((*I2* + *I2*) + (0 + *I1*))) + (0 + *I1*)))
     -
     (*I2* + *I1*)) - *I1*)
```

3.2 Eight Linearly Separable Tasks

The other experiments included the eight different supervised tasks used by Chalmers [3]. All of these tasks were linearly separable mappings. As shown in the following table they have five input units and one output unit. For each task twelve patterns are used. The task are at varying difficulty, for example the first one is to detect whether the fifth unit is on or off and the second task is to

learn to recognise a single specific pattern. The other tasks are more complex.

INPUTS						TASKS			
i1	i2	i3	i4	i5		T1	T2	T3	T4
1	1	1	1	1	-->	1	0	0	1
0	0	0	0	0	-->	0	0	1	0
0	1	1	1	0	-->	0	0	0	1
1	1	0	0	0	-->	0	1	1	1
1	0	1	0	1	-->	1	0	1	1
0	1	1	0	0	-->	0	0	1	0
0	1	1	1	1	-->	1	0	0	1
0	1	0	0	0	-->	0	0	1	0
1	1	0	0	1	-->	1	0	1	1
1	0	0	1	0	-->	0	0	1	1
1	0	1	1	0	-->	0	0	0	1
0	0	0	1	0	-->	0	0	1	0

The following are the result of evolving learning rules for the four tasks. The parameters of the genetic algorithms were the same but the percentage of variation in generating the expressions increased to 85 percent. Since the number of the input variables are higher (five) for these tasks, increasing the percentage of variation would force most, if not all, of the variables to be placed when expressions are produced randomly. Rather than using random integers 0 and 1 only in generating the expressions, real random numbers between 0 and 1 have also been tried (i.e. in Task 2 and Task 4). As it can be observed below this has increased the average success level of the evolved learning rules. Although it is not easy to describe their functions exactly within an expression, it must have facilitated the mapping of the evaluated values of the expressions to the target outputs, probably by somehow summarizing the values of the input pattern. In fact, overall success of evolved learning rules were just slightly below 100 percent which is higher than those found by Chalmers (around 90 percent) in his experiments.

TASK 1

```
1. ((((((*I3* * *I1*) + (0 + *I4*))
   +
   (((*I4* - *I5*) - (0 * *I2*)) - (1 - *I1*)))
```

```
      -
      ((*I2* * *I3*) + (*I4* + *I3*)))
      -
      ((((1 - *I1*) - (*I5* + *I5*)) + ((1 - *I5*) - (*I2* + *I3*)))
      +
      (0 + (*I3* - *I3*))))
      -
      (((0 + *I5*) + (0 * *I4*)) - (*I5* - *I4*)))
```

Success: 92 percent

2. ((*I2* * *I5*)

Success: 92 percent

```
3. (((((*I4* + *I2*) - ((*I2* * *I1*) + (*I5* + *I2*)))
      +
      (*I5* * *I3*)) + (((((((*I1* - *I4*) - (0 - *I5*))
      +
      (*I5* - *I4*)) + (*I2* + *I3*)) + (*I3* * *I5*))
      -
      ((*I2* - *I5*) - (((0 - *I4*) + (0 - *I1*))
      +
      (*I4* * *I3*)))) - (*I3* - *I5*)))
```

Success: 100 percent

```
4. ((*I2* + (*I5* - *I2*))
```

Success: 100 percent

TASK 2

```
1. ((((*I4* - *I4*) - ((*I5* + *I2*) + ((*I1* + *I1*)
      +
      (*I5* - *I3*)))) + (*I3* - *I2*))
      *
      (((*I4* - 0.056697) * ((*I2* + *I2*) + (*I4* + 0.991166)))
      *
      (*I2* - *I5*)))
```

Success: 100 percent

```
   2. (((*I1* - 0.49447) - ((*I4* + *I5*) - (*I2* + 0.588741)))
      -
      (*I2* + *I5*))

   3. ((((((*I1* - *I5*) + (*I3* + *I4*))
      -
      ((*I4* - 0.827728) + ((*I2* - *I5*) + (*I4* + 0.828348))))
      *
      (((*I3* - *I3*) + (*I1* - 0.416933))
      *
      ((*I2* + 0.128969) * (*I2* - 0.160986))))
      -
      (*I1* - *I5*))
      *
      (((*I3* - 0.260095) + (*I5* - *I2*)) + (*I1* - *I2*)))

   Success:  100 percent

   4. ((((*I4* - 0.354676) * (*I2* + 0.183204)) - 0.050929)
      *
      (((*I5* + 0.614033) * ((*I5* - 0.366376) * (*I5* + 0.14586)))
      -
      (*I2* + *I1*)))

   Success:  100 percent


TASK 3

   1. (((((1 - *I3*) + (1 + *I2*)) + ((*I4* + *I2*)
      +
      (*I1* - *I4*))) - (*I4* + *I2*)) - (*I2* + *I1*))

   Success:  100 percent

   2. (((((((*I1* - *I2*) - ((*I4* * *I5*) + (0 - *I2*)))
      +
      ((1 - *I5*) - ((*I3* - *I1*) - ((0 * *I2*)
      +
      (*I1* - *I3*))))) + (((*I1* * *I5*)
      -
      ((((*I1* - *I5*) + (((0 + *I4*) - (1 * *I1*)) + (1 * *I4*)))
      +
      (1 + *I5*)) + (0 - *I1*))) + (1 + *I2*)))
```

```
   +
   (*I2* - *I4*)) + (0 - *I5*))

Success:  92  percent

3. ((((((*I4* - *I2*) - ((*I3* + *I3*) - (1 + *I2*)))
    -
   ((*I5* * *I4*) + (0 * *I1*)))
   -
   (*I4* * *I3*)) + (*I1* + *I5*))

Success:  92  percent
```

In order to test how robust is the evolved learning rules for a given task, the rules have also been tested on unseen exemplars. For example, of all the tasks represented here, task four is the most difficult one. When tested on unseen patterns, the success of all of the four learning rules for this task remained at 100 percent.

```
TASK 4

1. (*I1* + ((*I4* + *I2*) + (*I5* - 0.893335)))

Success:  100 percent


2. ((*I1* + *I4*) * ((((*I4* + 0.350897) * (*I3* + 0.761547))
   *
   (((*I3* + 0.299322)+ ((((*I2* + *I1*) * (*I4* + *I4*))
   -
   (*I4* + *I2*)) - (*I3* + *I4*)))
   +
   (*I2* + *I2*))) + *I1*))

Success:  100 percent


3. (((*I1* + *I1*) - 0.781905)
   +
```

```
   (((*I2* + *I5*)
   *
   (((*I2* + 0.059971) * (*I3* + 0.113237)) * (*I2* - 0.331763)))
   +
   (((*I4* - 0.148709) * (*I1* + *I2*)) * (*I1* + *I4*))))


Success:  100 percent

4. ((*I1* - *I1*) - (((((*I4* + *I4*) - (*I4* - *I2*))
   *
   (((*I1* + *I1*) - (*I1* + *I4*)) + (*I4* + 0.28141)))
   -
   (*I1* + *I1*)) - (((*I4* + *I3*) * ((*I3* + *I5*)
   +
   ((*I5* + *I3*) * (*I2* - *I4*)))) * (*I4* + 0.30423))))


Success:  100 percent
```

3.3 Parity Problems

A difficult task for many learning algorithms are parity problems [20] where the required output is 1 if the input pattern contains an odd number of 1's; otherwise it is 0. This is a hard learning problem because very similar patterns (even different with one bit) may require completely different output. The XOR problem is one of the parity problems with size two. Although it took longer, expressions successfully evolved to code solution to XOR problem. For three bit parity problem it was difficult to evolve a learning rule with 100 percent fitness by using a population size of 30 (on the average 2 out of 10 runs would produce it). It would take around 15 generation to evolve a learning rule with 87.5 percent success (unsuccessful only on 1 out of 8 training pairs). Ninety percent of the times a solution found for the three bit parity problem with a minimum success level of 87.5 percent. The reason for this low level of performance is probably due to small population size and insufficient number of generations as compared to difficulty of the problem. However, it is clear that the encoding schema is capable of coding and finding a solution for this difficult task though the encoding schema involves as minimal as possible prior knowledge with respect to possible solutions of parity problems.

```
THREE BIT PARITY PROBLEM


  1. ((*I1* + ((((*I1* + *I2*) - (*I3* + *I1*)) + (*I2* - *I3*))
      *
     ((*I1* - 0.427487) * ((*I1* - *I2*)
      -
     ((*I2* - 0.526565) + (*I1* - 0.69849))))))

  Success: 100 percent


  2. ((*I1* - 0.306492)
     +
     (((((((*I2* + *I2*) - ((*I2* - *I2*) + (*I2* + *I2*)))
     *
     ((*I3* + *I1*) * ((*I1* - *I2*) + ((*I2* - 0.457992)
     +
     (((*I3* + 0.731704) * (*I2* + 0.775932))
     -
     ((*I3* - 0.70112) - (*I2* + *I2*)))))))
     +
     (*I3* - *I2*)) + (*I1* - *I2*)) * ((*I2* - *I3*)
     *
     ((*I3* - *I3*) - ((*I3* - *I3*) - (((*I2* - 0.773618)
     +
     (*I1* - 0.415343)) + (*I1* - 0.164109))))))) + 0.530177))

  Success: 100 percent
```

The above solution to parity problems exhibit a complex language. For the moment it is sufficient to observe that encoding schema is able to produce a solution for such difficult problems.

4 Conclusions

The experiments in this paper have shown that an encoding schema involving random expressions of input variables and random numbers can code and genetically evolve for learning rules of several supervised learning problems. It is

observed that using this encoding, evolution can provide a necessary basis for the learning rules to emerge, although any domain specific knowledge in coding the possible solutions are minimal. However, the research using this encoding is at its development stage and there are several issues that must be improved before going further.

As it has been shown in the previous section, it is possible that evolution can result in several different learning rules for the same task in hand. These solutions are sometimes very similar to each other, but are sometimes a totally different way of expressing the same solution. This provides promising evidence that the encoding schema would also be useful for more complex and larger problems since it can always find an alternative expression for a learning rule which is difficult to express and discover. Normally, we would like to have at least near optimal solutions without any redundant subexpressions. This can be accomplished by either (1) starting from simple elementary expressions and building up gradually or (2) eliminating redundant subexpressions from the final solutions. In the next experiments these issues will be of major concern.

The experiments shown here have been kept simple in order to show important characteristics of encoding schema and observe the evolution of learning rules. Only supervised learning tasks have been tested. Although experimenting with other learning methods is essential, the motivation of choosing supervised learning is directly related to the future aims of the research.

In the future, the encoding schema will be used to solve some hard supervised learning problems. These problems have been shown to be difficult to solve using conventional learning algorithms (i.e. back-propagation) due to the fact that the rule of learning contained in the target mapping may not refer to particular values of variables but rather it may refer to the possible relations among the input variables [22]. Thus the aim of the next experiments is to see whether the encoding strategy presented here would provide solution to hard learning problems under evolution. The next experiments will also involve target mappings with continuous rather than binary values and with more than one output values.

References

[1] D.H. Ackley and M.S. Litman. Interactions between learning and evolution. In Langton et al, editor, *Artificial Life II*. Addison-Wesley, 1992.

[2] R.K. Belew, J. McInerney, and N.N. Schraudolph. Evolving networks: using the genetic algorithms with connectionist learning. In Langton et al, editor, *Artificial Life II*. Santa Fe Institute, 1992.

[3] D.J. Chalmers. Evolution of learning: an experiment in genetic connectionism. In Touretzky et al, editor, *Connectionist Models*. Morgan Kaufmann, 1990.

[4] Ali Dasdan and Kemal Oflazer. Genetic syntesis of unsupervised learning algorithms. Technical report, Dept. of Coputer Engineering and Information Science, Bilkent University, 1994.

[5] A. DeJong, Kenneth. Genetic algorithms: a 10 years perspective. In *International Conference on Genetic Algorithms and their applications*. Lawrence Erlbaum Associates, 1985.

[6] A. DeJong, Kenneth. Learning with genetic algorithms: an overview. *Machine Learning*, 3(2):121–138, 1988.

[7] D. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Massacheusettes, 1989.

[8] I. Harvey. Adding species adaptation genetic algorithms:a basis for a conitinuing saga. Technical Report CSRP-221, University of Sussex, COGS, 1992a.

[9] I. Harvey. Evolutionary robotics and saga: The case for hill crowling and tournement selection. Technical Report CSRP-222, University of Sussex, COGS, 1992b.

[10] I. Harvey. The saga cross:the mechanics of recombination for species with variable-length genotypes. Technical Report CSRP-223, University of Sussex, COGS, 1992c.

[11] G.E. Hinton and S. J. Nowlan. How learning can guide evolution. *Complex systems*, 1:495–502, 1987.

[12] J. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, USA, 1975.

[13] J. Koza. *Genetic Programming:On the programming of computers by means of natural selection*. MIT press, 1992.

[14] I. Kuscu. Evolution of cooperation, MSc. Thesis, Dept. of AI, University of Edinburgh, 1991.

[15] Ibrahim Kuscu. Evolution of learning rules for supervised tasks ii: Hard learning problems. Technical Report CSRP-395, Uni. of Sussex, COGS, 1995.

[16] J. Maynard Smith. When learning guides evolution. *Nature*, 329:761–762, 1987.

[17] G.F. Miller and P.M. Todd. Exploring adaptive agency i:theory and methods for simulating the evolution of learning. In Touretzky et al, editor, *Connectionist Models*. Morgan Kaufmann, 1990.

[18] Melanie Mitchell and Stephanie Forrest. Genetic algorithms and artificial life. Technical Report 93-11-072, Santa Fe, 1993.

[19] S. Nolfi, J.L. Elman, and D. Parisi. Learning and evolution in neural networks. Technical Report CRL 9019, Uni. of California, 1990.

[20] D. Rumelhart, G. Hinton, and R. Williams. Learning internal representations by error propagation. In D. Rumelhart, J. McClelland, and the PDP Research Group, editors, *Parallel Distributed Processing: Explorations in the Micro-structures of Cognition. Vols I and II.* MIT Press, Cambridge, Mass., 1986a.

[21] S.S. Smith. A learning system based on genetic adaptive algorithms, phd. dissertation, university of pittsburgh, 1980.

[22] C. Thornton. Supervised learning of conditional approach: a case study. Technical Report 291, COGS, University of Sussex, 1993.

[23] D. Whitley. The genitor algorithm and why rank based-based allocation of reproductive trials is best. In J.D. Schaffer, editor, *Proceedings of Third iInternational Conference on Genetic Algorithms*, pages 116–123, 1989.